# Report
# Programming Class

*Programming Class 24/25*

Bachelor program UXD, THUAS


Student ID: 24097993

**Kerr Beeldens**

Picture by Albert Stoynov on Unsplash

X

# Test

# Contents

C1

## Chapter 1

# Introduction

This report contains my work from the programming class bootcamp. The report is split up into 7 chapters, with the first chapter (this one) being a short introduction and each of the following chapters is about 1 day of the bootcamp. These chapters are split up into a section with a summary of the topics of that day and a section that explains what the assignment was, what I tried to achieve and how you applied the topics from the "listen" segment. The final chapter is a reflection on the bootcamp as a whole. The .zip file that contains this report also contains the source code for each of the 5 assignments.

**Link to GitHub**

I am fully aware that no links are allowed and the .zip file is intended to stand on its own, but in case it is easier, the full contents are also available on GitHub through the following link:

https://github.com/KerrBeeldens/programming_class_24_25

C2

# Chapter 2

# Day 1: shapes, colors & variables

## 2.1 Summary of topics

During this lecture the layout of the PDE was explained. A basic program to draw a line was shown. Here, it was showed that one can draw basic shapes in processing using a variety of functions. These functions have a number of parameters. These are things like numbers, strings or booleans that change the behavior of the function. Examples are:

**Example 1.1**

```
println("Hello"); // print "Hello" in the console
size(400, 400); // set the screen size to 400 by 400 pixels
point(50, 90); // draw a point at x = 50, y = 90
```

It was also explained that by using two slashes, you can create comments, which help explain code to others (like I did above). Furthermore, the coordinate system of the screen (going from left to right and from above to below) and the drawing order was shown. Colors of objects can be changed, by using:

**Example 1.2**

```
fill(255, 0, 0); // set the fill color to red
background(0, 255, 0); // set the background to green
stroke(0, 0, 255, 127); // set the stroke color to translucent blue
```

The 4 parameters representing the R, G, B and Alpha values.
Next, variables where explained. These are small pieces of data that can represent numbers, letters, words, colors booleans etc. They are created using a type specifier like `int` or `boolean` followed by a name. Naming variables is done using camelCase. One can then use this name to assign data to the variable and then use the variable like:

**Example 1.3**

```
int a = 20; // declare a variable a and assign 20 to it
point(a, a); // use a to place a point at 20, 20
```

Processing also has build-in variables, like `width` and `height`.
Then, some basic mathematics where shown. You can add values or variables together using "+", subtract them using "-", multiply using "*" and divide using "/". It is important to keep track of the data types when doing this. It is also possible to change the datatype. For example, changing a value to an integer is possible using `int()`.

Finally the `random()` function was explained. It returns a float between 0 and the given number or when two numbers are given, between the two numbers, like this:

**Example 1.4**

```
1 float randomNumber = random(20); // return a random float from 0-20
2 int randomNumber = int(random(5, 20)); // return a random int from 5-19
```

## 2.2 Challenge description: half-tone

The goal of the assignment was to create a static artwork which is different, yet similar every time you run the program. My goal was to create a half-tone image. Half-tones are images consisting of little dots of various sizes, traditionally used in printing. The larger the dots, the darker that part of the image appears, creating a shading effect, as can be seen in Figuur 2.1.

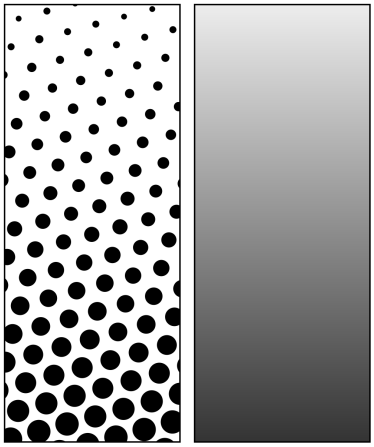

**Figure 2.1:** Half-tone (left) compared to the desired effect (right)

In the code, I load in multiple pieces of art and pick a random one that will be converted into a halftone. I also randomize the color of the dots using the `fill()` and `random()` functions. Throughout the program I use variables to store the image data and color values of a specific pixels in order to determine the size of the dots. I finally use the `ellipse()` function to draw the dots and achieve the effect.
The user can restart the application to get a new random image and dot color. I have included a total of se artworks which are displayed in Figuur 2.2.

(a) Johannes Vermeer - Girl with a Pearl Earring

(b) Edvard Munch - The Scream

(c) Leonardo da Vinci - Mona Lisa

(d) Jacques David - Napoleon Crossing The Alps

(e) Pablo Picasso - Autoritratto

(f) Rembrandt van Rijn - self portrait

(g) Vincent van Gogh - self portrait

**Figure 2.2:** Seven famous artworks as half-tones

C3

# Chapter 3

# Day 2: flow & conditionals

## 3.1 Summary of topics

The first topic of the lesson of today was about flow, which is the order in which the lines are executed. Yesterday's programs where executed from the top line to the bottom line, but this does not have to be the case. This can be achieved using the `setup()` and `draw()` methods. The `setup()` method is only ran once when the program launches. After that, the `draw()` method is repeatedly executed.
The second topic was about conditionals. A conditional takes the following form:

**example 2.1**

```
if (relational expression) {
    // lines to be executed if true
}
```

In here the relational expression is a statement which can be `true` or `false`. If the statement is true, the lines within the conditional are executed. Else, they are ignored. A relational statement takes the following form:

**example 2.2**

```
7 > 12 // > means greater than, so this is false
7 < 12 // > means less than, so this is true
7 == 12 // == means equal to, so this is false
7 != 12 // != means not equal to, so this is true
```

One can also combine add the = symbol to the less than or greater than symbols to mean less than or equal to/greater than or equal to. Next, it was also explained that with the `else` statement, you can tell the program what to do if a conditional is false:

**example 2.3**

```
if (relational expression) {
    // lines to be executed if true
} else {
    // lines to be executed if false
}
```

Two unrelated topics that where briefly mentioned today where those of indentation and the mouse location. Indentation is the convention to offset lines within blocks with a tab to the right. This makes

your program easier to read. The location of the mouse is stored in the variables `mouseX` and `mouseY`.

## 3.2 Challenge description: name

C4

# Chapter 4

# Day 3: conditionals & interactivity

## 4.1 Summary of topics

In the lesson of today we continued with conditionals. The `else` keyword was again explained, after which we also learned about the `else if` keyword. Expanding upon example 2.3 from yesterday we get:

**example 3.1**

```
if (relational expression 1) {
    // lines to be executed if expression 1 is true
} else if (relation expression 2) {
    // lines to be executed if expression 2 is true
} else {
    // lines to be executed if both expression 1 and 2 are false
}
```

Next, logical operators where demonstrated. These allow you to combine multiple relational expression together into a single relational expression. These are as follows:

**example 3.2**

```
expr 1 && expr 2 // Expr 1 and 2 need to be true to return true
expr 1 || expr 2 // Expr 1 and/or 2 need to be true to return true
!expr 1 // Expr 1 needs to be false to return true and vice versa
```

The next topic was about user interaction. First the variables `mouseX` and `mouseY` of yesterday were explained again. Next, two methods to detect mouse button presses and key presses where demonstrated. The first method uses the `keyPressed` and `mousePressed` variables. If a key or mouse button is pressed, the respective variable returns `true`. Then, with the variables `key` and `mouseButton`, you can read out which key or button was pressed. See the code examples below.

**example 3.3**

```
if (keyPressed) { // check if a key was pressed
    if (key == 'w') {
        // this code is executed when the 'w' key is pressed
    }
}
```

**example 3.4**

```
1 if (mousePressed) { // check if a mouse button was pressed
2     if (mouseButton == LEFT) {
3         // this code is executed when the left mouse button is pressed
4     }
5 }
```

The disadvantage of this first method is that the program is not interrupted when the user presses a key. If the main loop of the program takes a long time, a key press might be missed. The second method uses events, which interrupt the program if the conditions for that event are met (like pressing a button). A event triggers a method which is then executed once. Examples of mouse Events are `mousePressed()`, `mouseReleased()` and `mouseMoved()`. Examples of key events are `keyPressed()` and `keyReleased()`. In all these methods you can use the variables `key` and `mouseButton` similarly to example 3.3 and 3.4 to detect which key or button was pressed.

## 4.2 Challenge description: name

C5

# Chapter 5

# Day 4: loops & functions

## 5.1 Summary of topics

The topics for today where loops and functions. Loops make it possible to repeat a part of code multiple times, avoiding code duplication. There are two types of loops, while loops and for loops. The wile loop takes on the following structure:

**example 4.1**

```
1 while (statement) {
2     // code that will repeat as long as the statement is true
3 }
```

The while starts by checking if the statement is `true` or `false`. If the statement is `true` the code inside the block is executed once. The statement is checked again and while it remains `true`, the code is repeated. Important to note is that if the statement is false to begin with, the code is never executed and if the statement is always true, the program will get stuck in an endless loop.
Often times a variable is used which is updated inside of the loop. Eventually this will result in the statement becoming `false`. See example 4.2. The counter starts at 0, and inside the loop is increased by 1 each time until it becomes 10 at which point the statement will be false and the loop stops.

**example 4.1**

```
1 int counter = 0;
2 while (counter < 10) {
3     counter = counter + 1 // or counter++;
4 }
```

A for loop puts all of this logic together, making it better suited for this purpose. It takes on the following structure:

**example 4.3**

```
1 while (init; statement; update) {
2     // code that will repeat as long as the statement is true
3 }
```

Here, the `init` statement is executed first, then the `statement` is evaluated and if the test is true, the code inside the block is executed. After that, the `update` is executed and the process is repeated. In example 4.4 a real example is demonstrated. This code behaves exactly the same as example 4.3.

**example 4.4**

```
while (int counter = 0; counter < 10; counter++) {
    // code that will repeat as long as the statement is true
}
```

Finally it was also demonstrated that you can use loops within each other (nested loops). This can be used to fill a grid with dots for example.

The second topic where functions. These are self-contained modules that can be seen as an factory you give an instruction. The instructions are given through parameters. Some functions can also return a value using the `return` keyword. They are useful to make code easier to read, modify and expand. Let's take a look at two types:

**example 4.5**

```
void function(type1 parameter1, type2 parameter2) {
    // code that uses parameter1 and 2 to do something
}

int function(type1 parameter1, type2 parameter2) {
    // code that uses parameter1 and 2 to do something and return an integer
    return integerVariable;
}
```

The first function takes in two parameters and uses these to do something (but return nothing). The second function does the same, but will return an integer value which can be used further in the program. Both functions can be called using `function(parameter1, parameter2)`.

## 5.2 Challenge description: name

C6

# Chapter 6

# Day 5: arrays

## 6.1 Summary of topics

The topic of today were arrays. Arrays store a set of data elements under the same name. They can store all sorts of data types, but only one at the same time. They are especially useful for storing related data together, so you don't need to create lots of separate variables. There are a few different ways to create an array

**Example 5.1**

```
1 int[] data; // declare
2 data = new int[5]; // initialize
3 data[0] = 10; // assign
4
5 int[] data = new int[5]; // declare, initialize
6 data[0] = 0; // assign
7
8 int[] data = {10, 5, 7, 12, 20}; // initialize, declare, assign
```

You create an array using the datatype followed by square brackets and the variable name. Then, you can either use `new datatype[n]` to create an array of size `n` or use curly brackets followed by a number of variables separated by commas to immediately fill the array.
You can read a write data from the array by typing its name followed by square brackets. Inside these brackets you type the index of the value you want to get. The index is the position of the data in the array, starting at 0 for the first elements. You can write to an array by using the same method as before, together with the assign operator and the (new) value you want in the array. See example 5.2

**Example 5.2**

```
1 int[] data = {10, 5, 7, 12, 20};
2
3 data[3] // this will give the fourth value, 12
4 data[0] = 15; // set the first value (10) to 15
```

important to note that if the index is not in the array (like -1 or in the example, 5), you get an error.

The final topic was how you can use arrays in loops. Example 5.3 was shown during the lesson.

**Example 5.2**

```
float[] grades = {6, 6.5, 9, 8.5, 5};
float total = 0;

for(int i = 0; i < grades.length; i++){
    total += grades[i];
}
println(total/grades.length);
```

This code creates an array of grades and a variable `total` that will be used to store the sum of the grades. In a for loop the variable `i` starts at 0 and increases every iteration by 1. `grades.length` gets the length of the array, so the loop stops at the end of the array. Inside the loop, each value of the array is added to the `total` variable, which results in this variable being equal to the sum of the values in `grades`. Outside of the loop the average of the grades is printed.

## 6.2 Challenge description: name

C7

## Chapter 7

# Reflection